LA-UR-20-22060
Approved for public release; distribution is unlimited.

Title: Sparse MP4

Author(s): Wang, Daniel Anping
Strauss, Charles Shelby Murton
Springer, Jacob Mitchell
Thresher, Austin Morgan
Pritchard, Howard Porter Jr.
Kenyon, Garrett

Intended for: Southwest Symposium on Image Analysis and Interpretation (SSIAI), 2020-03-29 (Santa Fe, New Mexico, United States)

Issued: 2021-02-26 (rev.1)

# Sparse MP4


1st Daniel A. Wang
*Ultra-Scale Research Center*
*Los Alamos National Laboratory*
Los Alamos, USA
wang_a_daniel@lanl.gov

2nd Charles M. S. Strauss
*New Mexico Consortium*
Los Alamos, USA
charles.s.strauss@gmail.com

3nd Jacob M. Springer
*Swarthmore College*
Swarthmore, USA
jspring1@swarthmore.edu

4nd Austin Thresher
*Advanced Research in Cyber Systems*
*Los Alamos National Laboratory*
Los Alamos, USA
athresher@lanl.gov

5rd Howard Pritchard
*Ultra-Scale Research Center*
*Los Alamos National Laboratory*
Los Alamos, USA
hpritchard@lanl.gov

6th Garrett T. Kenyon
*Computation, Computing and Statistical Sciences*
*Los Alamos National Laboratory*
Los Alamos, USA
gkenyon@lanl.gov



***Abstract*—MP4 is currently the gold standard for video compression. Here we demonstrate that MP4 can be augmented by using a spatiotemporal sparse code optimized for the reconstruction of video portraits to up-sample data streams in which 75% of the pixels have been removed. Lossless MP4 can then be used to transmit the decimated video stream, yielding additional compression via a hybrid algorithm we refer to as Sparse MP4. We report that when compared with standard MP4 tuned for a similar overall compression ratio, Sparse MP4 achieves significantly higher PSNR and similar SSIM scores.**



**Keywords - Video Compression; Sparse Coding; Bottleneck Autoencoders; MP4**


## I. Introduction

Improvements in image and video compression draw significant commercial interest. Applications such as video calling software require efficient compression of spatiotemporal data in order to efficiently communicate information without unacceptable loss of quality. JPEG is a commonly used lossy format for image compression while MP4 is the international standard for compressing video images [1]. Bottleneck autoencoders can also perform lossy compression on images and videos [2] [3] [4] by learning a latent representation optimized for dimensional reduction while minimizing loss of content but may still lose perceptually important information [5].

Previous work shows that sparse coding supports improved compression of static image thumbnails compared to bottleneck autoencoders [6]. Sparse compression on static images works by removing every other pixel prior to transmission, employs a lossless compression algorithm to transmit the decimated image, and finally uses an overcomplete dictionary optimized for sparse reconstruction to infer the missing pixels upon reception. Here, we employ a similar strategy for compressing video portraits. The video stream is first decimated by removing every other pixel as well as by removing every other frame, yielding a raw compression ratio of 4:1. The decimated video stream can be further compressed by applying a lossless compression method, yielding a hybrid compression technique that will be referred to as Sparse MP4 throughout this paper. The missing pixels and frames are then inferred at the receiving end using a spatiotemporal dictionary optimized for the sparse reconstruction of video portraits. We report that when compared to lossy MP4 tuned to the same compression factor, Sparse MP4 achieves much higher PSNR values.

## II. What is Sparse Coding?

Sparse coding aims to infer an optimal representation of an input by inferring a solution that uses the fewest and presumably best features; the "best" features being the ones that capture high-order structure in the data. An ideal sparse representation ought to reduce the amount of trivially active neurons while simultaneously approaching zero reconstruction error. Here, sparse reconstructions are generated according to a Locally Competitive Algorithm [7], in which neurons compete via lateral inhibition. Once a sparse representation is inferred for a given input, the features are then refined according to a local Hebbian Learning Rule that reduces the reconstruction error given the sparse representation of that input. Formally, finding a sparse representation involves minimizing the following objective function in Equation 1:

$$E(\boldsymbol{I}, \boldsymbol{\phi}, \boldsymbol{a}) = \min_{\{\boldsymbol{a}, \boldsymbol{\phi}\}} [\, \frac{1}{2} ||\boldsymbol{I} - \boldsymbol{\phi}\boldsymbol{a}||^2 + \lambda ||\boldsymbol{a}||_p \,] \qquad (1)$$

$\boldsymbol{I}$ is a vector that represents an input image. Given an overcomplete dictionary of features $\boldsymbol{\phi}$, an ideal sparse representation would find a set of coefficients $\boldsymbol{a}$ that can most accurately reconstruct the input $\boldsymbol{I}$ while simultaneously minimizing a sparsity penalty. In the following, we assume the sparsity penalty is given by $||\boldsymbol{a}||_1$ although fractional norms are possible. $\lambda$ is a free parameter which controls the trade-off that occurs between sparsity and reconstruction error.

Stochastic gradient descent (SGD) is used to train an overcomplete dictionary initialized from a set of randomly generated features. Videos are sampled randomly, but for each video, frames were drawn in chronological order. This chronology helps to rapidly find sparse representations because the sparse representation of the new frames is initialized to

the sparse representation of the previous frames. For sparse coding, all frames were normalized to zero mean and unit standard deviation. The update rule for the feature kernels can be derived by taking the gradient of the cost function with respect to $\phi$ as shown by Equation 2:

$$\Delta\mathbf{\Phi} \propto -\frac{\partial E}{\partial \mathbf{\Phi}} = \mathbf{a} \otimes \{I - \mathbf{\Phi a}\} \tag{2}$$

equivalent to a local Hebbian update rule. Equation 2 was augmented by a momentum term to speed up convergence.

The overcomplete dictionary of feature kernels consisted of convolutional patches of size $18 \times 18$ spanning three consecutive frames with a spatial stride of 2 [8].

### A. The Dataset

The dataset used here for investigating compression quality of Sparse MP4 is a cropped Face Forensics dataset [9] of video portraits in which supplied bounding boxes were used to track and crop out the faces of each subject per video. A minimum square bounding box was determined that was sufficiently large to include every instance of a particular face in a given video portrait. Crops were then written as $128 \times 128$ individual frames at a target frame rate of 24 fps. A patch size of $18 \times 18$ pixels was used so that the network can only effectively learn local motion. Any large scale motion within the videos would completely pass outside the patches across multiple frames.

### B. PetaVision

To train a dictionary of spatiotemporal kernels for convolutional sparse coding and to generate sparse representations for each three frame sequence, we used PetaVision, a high-performance, open-source, neurosimulation toolbox optimized for sparse coding via LCA [10].

The basic set-up for inferring a sparse representation and learning a dictionary of convolutional spatiotemporal features using PetaVision is shown in Fig. 1. The input is passed to an error layer that encodes the difference, or residual, between the input and the reconstruction of that input. The error layer drives the LCA layer (V1) using the transpose of the weight matrix (green dashed line). The V1 layer, which encodes the sparse activation coefficients, has two roles. First, the LCA layer generates a reconstruction using a clone of the weight matrix (solid green line). Secondly, the LCA layer connects back to the error layer in order to update the weight matrix according to a local Hebbian rule (thick black line). This basic recurrent network can be applied to video simply by making multiple input and error layers and associated weight matrices, one for each video frame to be encoded.

### C. Compression by Sparse Coding

Sparse coding compression is achieved by decimating the input, transmitting the decimated video frames using a lossless compression algorithm, and then using a dictionary of spatiotemporal features optimized for sparse reconstruction to infer the missing information. Specifically, batches of three

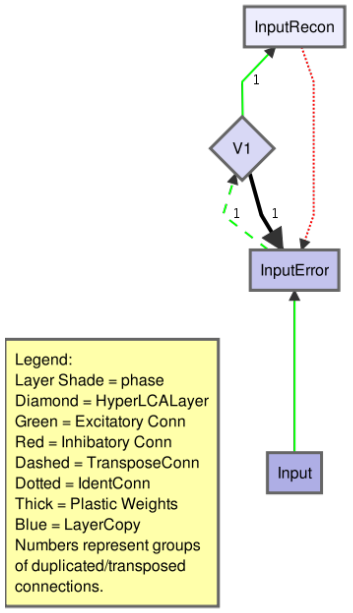


Fig. 1. The basic experimental set-up for an experiment run on PetaVision.

frames were passed into the network at a time. The data was down-sampled by first removing every other frame to achieve a 50% compression. A checkerboard mask was then applied to the remaining frames in order to remove every other remaining pixel. This additional 50% compression on the data produces a total rate of compression of 75% or a compression ratio of 4:1 on the original dataset. In order to recover the original input, a dictionary spanning three frames is trained on the decimated images. Although the network is set up to receive input from three frames, the second frame in each batch of three is removed due to the compression. Therefore in order to up-sample, the network must not only recover the missing pixels from the mask, but it must also infer the missing second frame in each batch. Once the dictionaries are sufficiently converged such that kernels no longer update during continued training, the network for up-sampling was run on a decimated validation set. The validation set is a cropped subset of Face Forensics videos that were not used for training. The validation reconstructions were evaluated to determine the quality of a sparse coding model for up-sampling and compressing video portraits.

Fig. 2 depicts an idealization of the steps necessary for compression and what a sparse coding reconstruction ought to look like given a masked caricatured input. The figure also shows ideal learned features given the caricatured input. Learned features hope to capture both local structure and local movement. Example features might include various circles to capture heads, eyes, and different curves to capture a moving mouth. The network, by using the best features to represent the smile and the frown found in the first and third frames, ought to infer a neutral mouth shape for the missing middle frame.

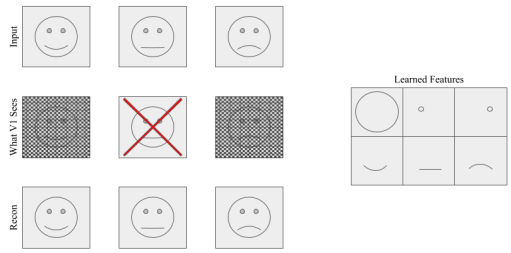


Fig. 2. Left: An idealized depiction of compression and reconstruction given a sparse coding network. In this depiction, sequences of three frames are taken from the input. A checkerboard mask is applied to each frame, and the center frame is complete removed. For each of the non-removed frames, the network must infer the missing pixels. For the missing frame, the network must totally interpolate the middle frame. Right: Idealized learned features given an input like the top row of the Left figure. A dictionary for this kind of input might learn these features: a circle for a head, smaller circles for eyes, and different curves representing a mouth that moves over time.

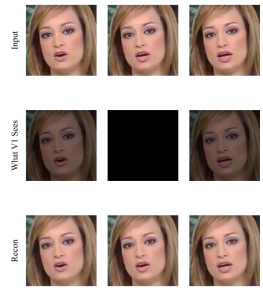


Fig. 3. Representative example of up-sampling via convolutional spatiotemporal sparse coding. Top row: there are three frames in chronological order cropped from a video portrait from the test set. Middle row: the input is then masked by a checkerboard mask and the middle frame is completely omitted. These three new frames are what the network actually sees. Bottom row: the network infers sparse reconstructions of the compressed input, from which the missing pixels and omitted frame can be in-painted.

Fig. 3 shows the compression and reconstructions of actual frames from the Face Forensics dataset that were passed into the network. Sequences of three frames were masked with the center frame being completely removed. The network then reconstructed the input by inferring missing pixels and interpolating the missing center frame. Moving feature kernels seem to successfully capture local physics as implied by Fig. 4.

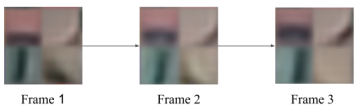


Fig. 4. A block of four features sampled from the three dictionaries for each frame in the up-sampling network. The four features shown in the blocks each move in different directions as the frames progress. This indicates that the network has learned local movement.

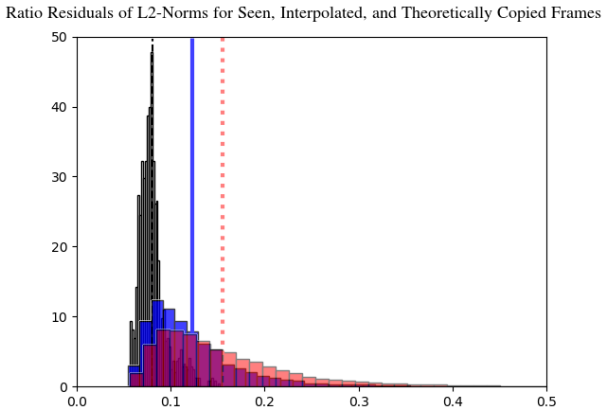


Fig. 5. Histograms of L2-norms of residuals from reconstructions of given frames *(black)*, interpolated frames *(blue)*, and copied reconstructions from the given frame prior to the interpolated frame *(orange)*. Each line is the mean of the histogram for its corresponding color.

## III. Up-Sampling Quality

In order to measure the quality of reconstructions produced by the up-sampling model, we made histograms of the calculated L2-norms of the residual errors. In the histograms represented by Fig. 5, the black histogram represents the residual error for the sparse reconstructions of decimated frames visible to the network. In the up-sampling model used, these visible decimated frames were frames one and three in each batch of three frames. As one can see, when the model can see the frame to be reconstructed, the residual error is very low.

The blue histogram represents the residual error from reconstructing the interpolated second frames. In each batch of three frames, this second frame was completely removed and had to be entirely inferred. At a glance, it seems that interpolation is working well. A third orange histogram was plotted to determine what the residual error would look like if the reconstruction for the previous frame was used as the interpolation. That is to say, if the network could just take the reconstructions of either frame one or frame three, pass them off as an ”interpolation”, and still do better than our truly interpolated frames; the interpolation indicated by the blue histogram would not be meaningful. After plotting the residual error from copying adjacent frames for interpolation, it can be seen that the true interpolation is noticeably better than copying. What this means is that the up-sampling network is not simply copying adjacent frames, spatial up-sampling is working and the network is successfully finding a reconstruction that is temporally ”between” the given frames.

## IV. Compression Quality

The operating hypothesis being tested here is that compression of natural videos by sparse coding outperforms the compression quality of both a bottleneck autoencoder [11] [12] as well as standard MP4 compression. We use the standard

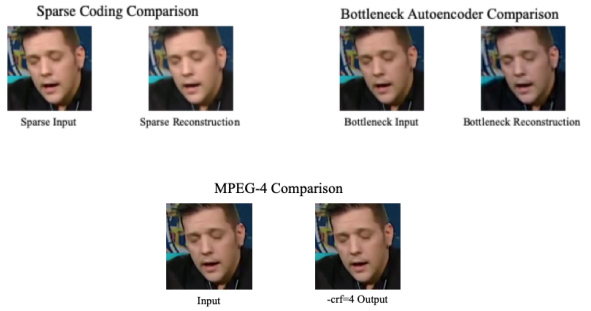


Fig. 6. *(Top Left)* Sparse reconstruction and *(Top Right)* bottleneck autoencoder reconstruction compared compared to original input frame. *(Bottom)* A frame from an MP4 video compressed at -crf = 4 is shown next to the original frame.

metrics for compression quality PSNR and SSIM to measure the quality of each compression scheme. While PSNR uses the absolute error of individual pixels, SSIM samples the structure, contrast, and luminance of the images to produce a more perceptually-based measurement [13]. For both PSNR and SSIM, a higher score corresponds to better compression.

We compared the reconstructions inferred from the decimated video to the input frames using both PSNR and SSIM. We also applied the same compression metrics to the output of a three layer denoising bottleneck autoencoder optimized on the same training data. Because input images were compressed by 75% for the sparse coding network, the bottleneck autoencoder was also set to compress its input at a 75% rate. The code used to compress frames into an MP4 format at lossless or lossy rates is an open-source project called FFmpeg [14]. Images were converted into an .mp4 format with FFmpeg [14] by adjusting values of a parameter called -crf to control for reconstruction quality and degree of compression. A -crf value of 0 corresponds to lossless compression while higher values correspond with increasingly lossy compression. Example reconstructions from sparse coding, the bottleneck autoencoder, and from MP4 are shown in Fig. 6

We obtained quantitative comparisons between compression methods as follows: Starting with a set of PNG frames extracted at 24fps from each video portrait in the validation set, we used lossless MP4 compression to determine a baseline compression ratio. Then, Sparse MP4 was applied to the same set of videos, yielding an average compression factor of 2.8422 to 2.8482. By adjusting the -crf variable when compressing undecimated frames into MP4 videos with standalone MP4, -crf values of 3 and 4 produce compression factors of 2.6330 and 2.9941 respectively. Videos produced from these values were thus compared against Sparse MP4 to test for compression quality.

When PSNR was applied to the reconstructions of the sparse coding network in Table. I, the PSNR values ranged from an average of 77.0338 for the second (interpolated) frames to 79.0245 for the first (masked) frames. PSNR applied to the bottleneck autoencoder reconstructions produced values that had an average range between 31.4293 for the first frames and 32.4929 for the second frames. When SSIM was applied to the reconstructions from the bottleneck autoencoder in Table. II, the lowest average SSIM value was 0.9841 for frame two and the highest was 0.9929 for frame three. Similarly, the lowest average value for the bottleneck autoencoder was 0.9429 for frame three and the highest was 0.9472 for frame two.

TABLE I

| PSNR Values - Scale 0-100 | | | |
|---|---|---|---|
| Methods | Frame 1 | Frame 2 | Frame 3 |
| Sparse Coding | 79.0245 | 77.0338 | 78.6417 |
| Bottleneck Autoencoder | 31.4293 | 32.4929 | 32.0486 |
| | -crf = 3 | -crf = 4 | |
| MP4 | 45.9641 | 45.4548 | |

TABLE II

| SSIM Values - Scale 0-1 | | | |
|---|---|---|---|
| Methods | Frame 1 | Frame 2 | Frame 3 |
| Sparse Coding | 0.9923 | 0.9841 | 0.9929 |
| Bottleneck Autoencoder | 0.9430 | 0.9472 | 0.9429 |
| | -crf = 3 | -crf = 4 | |
| MP4 | 0.9930 | 0.9923 | |

For both PSNR and SSIM, the second frame for the sparse coding model had the lowest average quality. It makes sense that the second frame of the sparse coding network should have lower values when compared against frames one and three. This is because the sparse coding method of compression completely removes the middle frame and thus the network is required to entirely interpolate the middle frame whereas the adjacent frames only need to infer every other pixel. With PSNR there can be seen a significant discrepancy in compression quality between sparse coding and the bottleneck autoencoder with which the sparse coding model significantly outperforms the bottleneck autoencoder (Table I). Conversely, although SSIM indicates that sparse coding still outperforms the bottleneck autoencoder (Table II), the difference is not as significant as with PSNR.

PSNR was applied on videos compressed at -crf values of 3 and 4 and returned average values of roughly 45.9641 and 45.4548 respectively (Table I). On the other hand, SSIM produced respective average values of 0.9930 and 0.9923 (Table II). When compared against the PSNR and SSIM values produced by sparse coding, sparse coding seems to perform similarly well according to SSIM, but PSNR indicates that sparse coding is able to outperform MP4 and produce higher quality reconstructions for the same degree of compression.

## V. Conclusion

This paper substantiates the hypothesis that video compression based on sparse coding can outperform the compression provided by denoising bottleneck autoencoders according to standard compression metrics PSNR and SSIM. Also, when Sparse MP4 and standalone MP4 compression are compared at the same rate of compression, Sparse MP4 can match the compression quality of standalone MP4 in SSIM and in the case of PSNR, can even outperform the state-of-the-art.

## REFERENCES


[1] T. Gloe, A. Fischer, and M. Kirchner, "Forensic analysis of video file formats," *Digital Investigation*, vol. 11, pp. S68–S76, 2014.

[2] J. Pessoa, H. Aidos, P. Tomás, and M. A. T. Figueiredo, "End-to-end learning of video compression using spatio-temporal autoencoders," 2019. [Online]. Available: https://openreview.net/forum?id=HyllasActm

[3] S. Ma, X. Zhang, C. Jia, Z. Zhao, S. Wang, and S. Wanga, "Image and video compression with neural networks: A review," *IEEE Transactions on Circuits and Systems for Video Technology*, 2019.

[4] D. Liu, Y. Li, J. Lin, H. Li, and F. Wu, "Deep learning-based video coding: A review and a case study," *arXiv preprint arXiv:1904.12462*, 2019.

[5] H. Zenil, N. A. Kiani, and J. Tegnér, "Quantifying loss of information in network-based dimensionality reduction techniques," *Journal of Complex Networks*, vol. 4, no. 3, pp. 342–362, 2015.

[6] Y. Watkins, M. Sayeh, O. Iaroshenko, and G. T. Kenyon, "Image compression: Sparse coding vs. bottleneck autoencoders," *CoRR*, vol. abs/1710.09926, 2017. [Online]. Available: http://arxiv.org/abs/1710.09926

[7] C. J. Rozell, D. H. Johnson, R. G. Baraniuk, and B. A. Olshausen, "Sparse coding via thresholding and local competition in neural circuits," *Neural computation*, vol. 20, no. 10, pp. 2526–2563, 2008.

[8] P. F. Schultz, D. M. Paiton, W. Lu, and G. T. Kenyon, "Replicating kernels with a short stride allows sparse reconstructions with fewer independent kernels," *arXiv preprint arXiv:1406.4205*, 2014.

[9] A. Rössler, D. Cozzolino, L. Verdoliva, C. Riess, J. Thies, and M. Nießner, "Faceforensics: A large-scale video dataset for forgery detection in human faces," *CoRR*, vol. abs/1803.09179, 2018. [Online]. Available: http://arxiv.org/abs/1803.09179

[10] 2019. [Online]. Available: https://github.com/PetaVision

[11] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *science*, vol. 313, no. 5786, pp. 504–507, 2006.

[12] Y. Z. Watkins and M. R. Sayeh, "Image data compression and noisy channel error correction using deep neural network," *Procedia Computer Science*, vol. 95, pp. 145–152, 2016.

[13] A. Hore and D. Ziou, "Image quality metrics: Psnr vs. ssim," in *2010 20th International Conference on Pattern Recognition*. IEEE, 2010, pp. 2366–2369.

[14] FFmpeg Team, "Ffmpeg." [Online]. Available: https://github.com/FFmpeg/FFmpeg